Nazwa implementacji: Przechowywanie danych

Autor: michal.czyzewski

Opis implementacji:

Aplikację ToDo uzupełnimy o możliwość trwałego zapisania wprowadzonych informacji i ich odczytu przy ponownym uruchomieniu strony.

Nasza aktualna aplikacja pozwala nam na dodawanie zadań do listy z różnymi priorytetami i odznaczanie tych, które mamy zakończone. Problem pojawia się, gdy zamykamy przeglądarkę lub odświeżymy stronę. Wszystkie wprowadzone informacje znikają i korzystanie z takiej aplikacji zupełnie nie ma sensu.

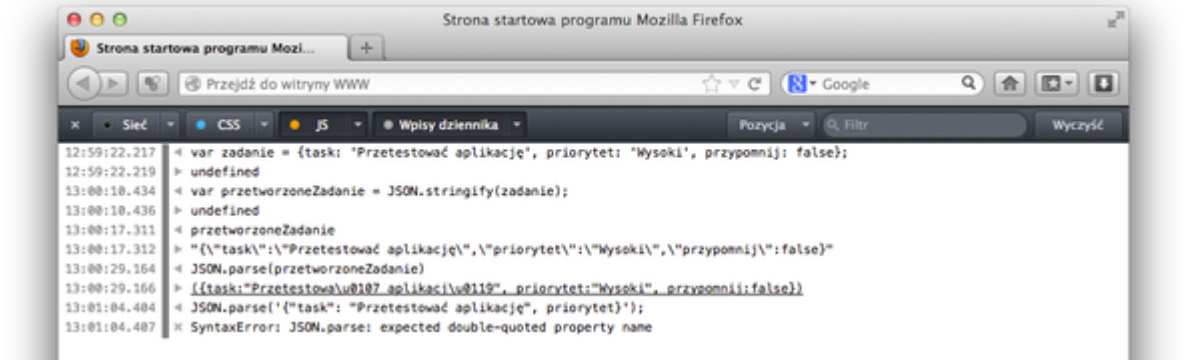
JSON – podstawowy format danych

Zanim przejdziemy do sposobów zapisywania danych w przeglądarce, powinniśmy zapoznać się z powszechnie używanym w aplikacjach internetowych formatem danych. Opiera się on mocno na składni języka JavaScript, dzięki czemu jest czytelny dla każdej osoby, która poznała podstawy tego języka. Do zamieniania obiektów i tablic na format JSON, korzystamy z następujących metod:

JSON.stringify() – zamienia wartość zmiennej podanej w parametrze na łańcuch znaków w formacie JSON.

- JSON.parse() – zamienia łańcuch znaków zachowujący poprawny format JSON na tablicę lub obiekt w JavaScript.

Zadanie: W konsoli przeglądarki utwórz dowolny obiekt lub tablice zawierającą kilka elementów. Sprawdź, jak będzie wyglądał zapis tego obiektu w formacie JSON. Sprawdź, czy po przetworzeniu JSON z powrotem do obiektu, jego struktura się zmieniła. Spróbuj do JSON.parse() podać łańcuch znaków zawierający błędy w formatowaniu. Co zostanie zwrócone?



Zapis i odczyt w sessionStorage i localStorage

Do przechowywania prostych struktur danych możemy skorzystać z prostych rozwiązań sessionStorage i localStorage. Zapis i odczyt danych w nich odbywa się w ten sam sposób, a jedyna różnica pomiędzy nimi to, że dane w sessionStorage są usuwane przy zamknięciu okna przeglądarki, ale nie odświeżeniu strony. Natomiast localStorage przechowuje zapisane informacje tak długo, aż nie zostaną usunięte przez operacje w programie. Oba obiekty mają dwie proste metody:

- **setItem('klucz', wartosc) – zapisuje wartość przechowywaną w zmiennej wartosc pod adresem podanym jako pierwszy parametr, czyli 'klucz'.**

getITem('klucz') – pobiera wartość zapisana w localStorage pod podanym adresem.

Uwaga: Każdy element zapisywany do sessionStorage lub localStorage jest przetwarzany do łańcucha znaków, więc w momencie wywołania otrzymamy:

```
localStorage.setItem('zadanie', {elem: 'Test'});
```


KAPITAŁ LUDZKI NARODOWA STRATEGIA SPÓJNOŚCI
ORE
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Projekt “Strategia Wolnych i Otwartych Implementacji jako innowacyjny model zainteresowania kierunkami informatyczno-technicznymi oraz wspierania uczniów i uczennic w kształtowaniu kompetencji kluczowych” współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

```
localStorage.getItem('zadanie'); // "[object Object]"
```

Żeby poprawnie zapisywać takie dane jak tablice lub całe obiekty potrzebujemy zamienić je na łańcuch znaków w formacie JSON:

```
var zadanie = {elem: 'Test'};
localStorage.setItem('zadanie', JSON.stringify(zadanie));
JSON.parse(localStorage.getItem('zadanie'));
```

Zadanie: W konsoli przeglądarki przetestuj zapisywanie danych do sessionStorage i localStorage, oraz sprawdź co dzieje się z zapisanymi danymi przy odświerzeniu strony i ponownym uruchomieniu przeglądarki.

Integracja aplikacji ToDo z sessionStorage i localStorage

Zadanie: Stwórz oddzielny plik storage.js i dodaj w nim obsługę zdarzenia submit na <form> gdzie dane wprowadzone w formularzu będą dodane do tablicy zawierającej listę wszystkich zadań wyświetlonych na liście, a następnie przetworzone do formatu JSON i zapisane w localStorage.

Przy załadowaniu strony lista zadań powinna być ponownie pobrana z localStorage, a elementy w niej zawarte dodane w postaci listy do dokumentu HTML.

Operacje na bazie danych IndexedDB

localStorage jest bardzo prosty w obsłudze i nie wymaga dużej ilości kodu, jednak w przypadku większych ilości danych, gdy na przykład nasza lista zadań urośnie do kilkuset elementów, będzie bardzo ograniczał szybkość działania aplikacji, oraz nie da możliwości wygodnego wyszukiwania, oraz modyfikowania pojedynczych zadań. Za każdym razem musimy pobierać i zapisywać całą zawartość.

Gdy potrzebujemy bardziej zaawansowanych operacji, warto wykorzystać kolejne narzędzie dostępne w przeglądarkach wspierających HTML5: indexedDB.

Tworzenie i otwieranie bazy danych

Żeby skorzystać z indexedDB, musimy najpierw utworzyć ja pod wybraną nazwą:

```
var db;

var request = indexedDB.open('ToDo', 1);

request.onerror = function(event) {

alert('Wystąpił błąd! Nie można utworzyć bazy danych.);

};


request.onsuccess = function(event) {

db = request.result;

};


request.onupgradeneeded = function(event) {

request.result.createObjectStore('tasks');
```


KAPITAŁ LUDZKI NARODOWA STRATEGIA SPÓJNOŚCI ORE FWIOO UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Projekt “Strategia Wolnych i Otwartych Implementacji jako innowacyjny model zainteresowania kierunkami informatyczno-technicznymi oraz wspierania uczniów i uczennic w kształtowaniu kompetencji kluczowych” współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

```
};
```

W ten sposób mamy utworzoną bazę danych o nazwie ToDo, dostępną pod zmienną db. Gdy operacja otwierania lub tworzenia bazy danych się nie powiedzie, użytkownik otrzymał komunikat o błędzie. onupgradeneeded jest uruchamiane, kiedy baza danych nie istnieje lub ma inny numer wersji, niż podany jako drugi parametr w indexedDB.open().

Dodawanie elementów do bazy

Żeby dodać element do bazy danych, musimy najpierw utworzyć transakcję, wskazując, na jakich elementach będziemy dokonywać zmiany, oraz czy tranzakcja będzie tylko odczytywała, czy też zapisywała rekordy.

```
var trans = db.transaction(['tasks'], 'readwrite');
```

W ten sposób tworzymy tranzakcję dokonującej zmian na elementach tasks. Tranzakcje domyślnie są ustawione na read, więc jeśli planujemy wprowadzać zmiany do bazy, należy podać 2 argument o wartości readwrite.

Następnie na utworzonej tranzakcji wywołujemy metodę objectStore() podając jako parametr jeden z typów obiektów, na których będziemy dokonywać zmiany. W naszym przypadku jest to tylko tasks.

```
var store = trans.objectStore('tasks');
```

Na tak utworzonym obiekcie możemy wywoływać metody pozwalające na dodawanie, pobieranie i aktualizowanie danych potrzebnych w naszej aplikacji:

- add() – dodaje nowy element do bazy. Uwaga: Jeśli element już istnieje wa bazie, zostanie zwrócony komunikat o błędzie.
- put() – tak jak add() dodaje element do bazy, ale jeśli już istnieje w bazie, jego wartość zostanie zaktualizowana.
- delete() – usuwa podany element z bazy danych.
- clear() – usuwa wszystkie elementy z bazy danych.
- count() – zwraca liczbę elementów zapisanych w bazie danych.
- get() – pobiera wybrany element z bazy danych.
- openCursor() – tworzy obiekt, który pozwala na pobranie po kolei wszystkich elementów z bazy danych.

Zadanie: Stwórz oddzielny plik database.js i tak samo jak w przypadku storage.js napisz obsługę zdarzeń, które będą dodawały i aktualizowały informacje o liście zadań w bazie danych utworzonej poprzez indexedDB.

KAPITAŁ LUDZKI NARODOWA STRATEGIA SPÓJNOŚCI
ORE
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Projekt “Strategia Wolnych i Otwartych Implementacji jako innowacyjny model zainteresowania kierunkami informatyczno-technicznymi oraz wspierania uczniów i uczennic w kształtowaniu kompetencji kluczowych” współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.